# React Integration Guide for Bilingual API

This guide shows how to integrate the bilingual API with a React frontend, supporting both English and Arabic content.

## Overview

The API now supports bilingual content (English and Arabic) for the following models:

- **Products**: `name`, `description`, `category`, `unit`, `customFields`
- **Addresses**: `addressLine1`, `addressLine2`, `city`, `state`, `country`
- **Categories**: `name`, `description`
- **Orders**: `status` (localized in responses)
- **Reviews**: `comment`

## Language Management

### 1. Language Context

```
// contexts/LanguageContext.js
import React, { createContext, useContext, useState } from 'react';

const LanguageContext = createContext();

export const LanguageProvider = ({ children }) => {
  const [currentLanguage, setCurrentLanguage] = useState('en');

  const toggleLanguage = () => {
    setCurrentLanguage(prev => prev === 'en' ? 'ar' : 'en');
  };

  return (
    <LanguageContext.Provider value={{ currentLanguage, toggleLanguage }}>
      {children}
    </LanguageContext.Provider>
  );
};

export const useLanguage = () => {
  const context = useContext(LanguageContext);
  if (!context) {
    throw new Error('useLanguage must be used within a LanguageProvider');
  }
  return context;
};
```

### 2. Language Toggle Component

```
// components/LanguageToggle.js
import React from 'react';
import { useLanguage } from '../contexts/LanguageContext';

const LanguageToggle = () => {
  const { currentLanguage, toggleLanguage } = useLanguage();

  return (
    <button onClick={toggleLanguage}>
      {currentLanguage === 'en' ? 'العربية' : 'English'}
    </button>
  );
};

export default LanguageToggle;
```

## API Integration

### 1. API Service

```
// services/api.js
const API_BASE_URL = process.env.REACT_APP_API_URL || 'http://localhost:3000/api';

class ApiService {
  constructor() {
    this.baseURL = API_BASE_URL;
  }

  // Helper to get language parameter
  getLanguageParam(language) {
    return language ? `?lang=${language}` : '';
  }

  // Generic request method
  async request(endpoint, options = {}) {
    const token = localStorage.getItem('token');
    const headers = {
      'Content-Type': 'application/json',
      ...(token && { Authorization: `Bearer ${token}` }),
      ...options.headers,
    };

    const response = await fetch(`${this.baseURL}${endpoint}`, {
      ...options,
      headers,
    });

    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }
    return response.json();
  }

  // Product APIs
  async getProducts(language = 'en') {
    return this.request(`/products${this.getLanguageParam(language)}`);
  }

  async createProduct(productData) {
    return this.request('/products/addProductsByBusiness', {
      method: 'POST',
      body: JSON.stringify(productData),
    });
  }

  // Address APIs
  async getAddresses(language = 'en') {
    return this.request(`/addresses${this.getLanguageParam(language)}`);
  }

  async createAddress(addressData) {
    return this.request('/addresses', {
      method: 'POST',
      body: JSON.stringify(addressData),
    });
  }

  async updateAddress(id, addressData) {
    return this.request(`/addresses/${id}`, {
      method: 'PUT',
      body: JSON.stringify(addressData),
    });
  }

  // Category APIs
  async getCategories(language = 'en') {
    return this.request(`/categories${this.getLanguageParam(language)}`);
  }

  async createCategory(categoryData) {
    return this.request('/categories', {
      method: 'POST',
      body: JSON.stringify(categoryData),
    });
  }

  // Order APIs
  async getMyOrders(language = 'en') {
    return this.request(`/orders/my${this.getLanguageParam(language)}`);
  }

  async createOrder(orderData) {
    return this.request('/orders', {
      method: 'POST',
      body: JSON.stringify(orderData),
    });
  }

  // Review APIs
  async getProductReviews(productId, language = 'en') {
    return
this.request(`/reviews/products/${productId}/reviews${this.getLanguageParam(langua
ge)}`);
  }

  async createReview(productId, reviewData) {
    return this.request(`/reviews/products/${productId}/reviews`, {
      method: 'POST',
      body: JSON.stringify(reviewData),
    });
  }
}

export default new ApiService();
```

## Form Components

### 1. Product Creation Form

```
// components/ProductForm.js
import React, { useState } from 'react';
import { useLanguage } from '../contexts/LanguageContext';
import apiService from '../services/api';

const ProductForm = () => {
  const { currentLanguage } = useLanguage();
  const [productData, setProductData] = useState({
    category: { en: '', ar: '' },
    name: { en: '', ar: '' },
    description: { en: '', ar: '' },
    unit: { en: '', ar: '' },
    customFields: [
      { key: { en: '', ar: '' }, value: { en: '', ar: '' } }
    ]
  });

  const handleCategoryChange = (language, value) => {
    setProductData(prev => ({
      ...prev,
      category: { ...prev.category, [language]: value }
    }));
  };

  const handleNameChange = (language, value) => {
    setProductData(prev => ({
      ...prev,
      name: { ...prev.name, [language]: value }
    }));
  };

  const handleDescriptionChange = (language, value) => {
    setProductData(prev => ({
      ...prev,
      description: { ...prev.description, [language]: value }
    }));
  };

  const handleUnitChange = (language, value) => {
    setProductData(prev => ({
      ...prev,
      unit: { ...prev.unit, [language]: value }
    }));
  };

  const handleCustomFieldChange = (index, field, language, value) => {
    setProductData(prev => ({
      ...prev,
      customFields: prev.customFields.map((item, i) =>
        i === index
          ? { ...item, [field]: { ...item[field], [language]: value } }
          : item
      )
    }));
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const response = await apiService.createProduct(productData);
      console.log('Product created:', response);
    } catch (error) {
      console.error('Error creating product:', error);
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <div>
        <label>Category:</label>
        <div>
          <input
            type="text"
            placeholder="English Category"
            value={productData.category.en}
            onChange={(e) => handleCategoryChange('en', e.target.value)}
          />
          <input
            type="text"
```

```
              placeholder="Arabic Category"
              value={productData.category.ar}
              onChange={(e) => handleCategoryChange('ar', e.target.value)}
            />
          </div>
        </div>

        <div>
          <label>Name:</label>
          <div>
            <input
              type="text"
              placeholder="English Name"
              value={productData.name.en}
              onChange={(e) => handleNameChange('en', e.target.value)}
            />
            <input
              type="text"
              placeholder="Arabic Name"
              value={productData.name.ar}
              onChange={(e) => handleNameChange('ar', e.target.value)}
            />
          </div>
        </div>

        <div>
          <label>Description:</label>
          <div>
            <textarea
              placeholder="English Description"
              value={productData.description.en}
              onChange={(e) => handleDescriptionChange('en', e.target.value)}
            />
            <textarea
              placeholder="Arabic Description"
              value={productData.description.ar}
              onChange={(e) => handleDescriptionChange('ar', e.target.value)}
            />
          </div>
        </div>

        <div>
          <label>Unit:</label>
          <div>
            <input
              type="text"
              placeholder="English Unit"
              value={productData.unit.en}
              onChange={(e) => handleUnitChange('en', e.target.value)}
            />
            <input
              type="text"
              placeholder="Arabic Unit"
              value={productData.unit.ar}
              onChange={(e) => handleUnitChange('ar', e.target.value)}
            />
          </div>
        </div>

        {/* Custom Fields */}
        {productData.customFields.map((field, index) => (
          <div key={index}>
            <label>Custom Field {index + 1}:</label>
            <div>
              <input
                type="text"
                placeholder="English Key"
                value={field.key.en}
                onChange={(e) => handleCustomFieldChange(index, 'key', 'en',
e.target.value)}
              />
              <input
                type="text"
                placeholder="Arabic Key"
                value={field.key.ar}
                onChange={(e) => handleCustomFieldChange(index, 'key', 'ar',
e.target.value)}
              />
              <input
                type="text"
                placeholder="English Value"
                value={field.value.en}
                onChange={(e) => handleCustomFieldChange(index, 'value', 'en',
e.target.value)}
              />
              <input
                type="text"
                placeholder="Arabic Value"
                value={field.value.ar}
                onChange={(e) => handleCustomFieldChange(index, 'value', 'ar',
e.target.value)}
              />
            </div>
          </div>
        ))}

        <button type="submit">Create Product</button>
      </form>
    );
};

export default ProductForm;
```

### 2. Address Form

```
// components/AddressForm.js
import React, { useState } from 'react';
import { useLanguage } from '../contexts/LanguageContext';
import apiService from '../services/api';

const AddressForm = () => {
  const { currentLanguage } = useLanguage();
  const [addressData, setAddressData] = useState({
    addressLine1: { en: '', ar: '' },
    addressLine2: { en: '', ar: '' },
    city: { en: '', ar: '' },
    state: { en: '', ar: '' },
    postalCode: '',
    country: { en: '', ar: '' }
  });

  const handleFieldChange = (field, language, value) => {
    setAddressData(prev => ({
      ...prev,
      [field]: { ...prev[field], [language]: value }
    }));
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const response = await apiService.createAddress(addressData);
      console.log('Address created:', response);
    } catch (error) {
      console.error('Error creating address:', error);
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <div>
        <label>Address Line 1:</label>
        <div>
          <input
            type="text"
            placeholder="English Address Line 1"
            value={addressData.addressLine1.en}
            onChange={(e) => handleFieldChange('addressLine1', 'en',
e.target.value)}
          />
          <input
            type="text"
            placeholder="Arabic Address Line 1"
            value={addressData.addressLine1.ar}
            onChange={(e) => handleFieldChange('addressLine1', 'ar',
e.target.value)}
          />
        </div>
      </div>

      <div>
        <label>Address Line 2:</label>
        <div>
          <input
            type="text"
            placeholder="English Address Line 2"
            value={addressData.addressLine2.en}
            onChange={(e) => handleFieldChange('addressLine2', 'en',
e.target.value)}
          />
          <input
            type="text"
            placeholder="Arabic Address Line 2"
            value={addressData.addressLine2.ar}
            onChange={(e) => handleFieldChange('addressLine2', 'ar',
e.target.value)}
          />
        </div>
      </div>

      <div>
        <label>City:</label>
        <div>
          <input
            type="text"
            placeholder="English City"
            value={addressData.city.en}
            onChange={(e) => handleFieldChange('city', 'en', e.target.value)}
          />
          <input
            type="text"
            placeholder="Arabic City"
            value={addressData.city.ar}
            onChange={(e) => handleFieldChange('city', 'ar', e.target.value)}
          />
        </div>
      </div>

      <div>
        <label>State:</label>
        <div>
          <input
            type="text"
            placeholder="English State"
            value={addressData.state.en}
            onChange={(e) => handleFieldChange('state', 'en', e.target.value)}
          />
          <input
            type="text"
            placeholder="Arabic State"
            value={addressData.state.ar}
            onChange={(e) => handleFieldChange('state', 'ar', e.target.value)}
          />
        </div>
      </div>

      <div>
        <label>Postal Code:</label>
        <input
          type="text"
          value={addressData.postalCode}
          onChange={(e) => setAddressData(prev => ({ ...prev, postalCode:
e.target.value }))}
        />
      </div>

      <div>
        <label>Country:</label>
        <div>
          <input
            type="text"
            placeholder="English Country"
            value={addressData.country.en}
            onChange={(e) => handleFieldChange('country', 'en', e.target.value)}
          />
          <input
            type="text"
            placeholder="Arabic Country"
            value={addressData.country.ar}
            onChange={(e) => handleFieldChange('country', 'ar', e.target.value)}
          />
        </div>
      </div>

      <button type="submit">Create Address</button>
    </form>
  );
};

export default AddressForm;
```

### 3. Category Form

```
// components/CategoryForm.js
import React, { useState } from 'react';
import { useLanguage } from '../contexts/LanguageContext';
import apiService from '../services/api';

const CategoryForm = () => {
  const { currentLanguage } = useLanguage();
  const [categoryData, setCategoryData] = useState({
    name: { en: '', ar: '' },
    description: { en: '', ar: '' }
  });

  const handleFieldChange = (field, language, value) => {
    setCategoryData(prev => ({
      ...prev,
      [field]: { ...prev[field], [language]: value }
    }));
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const response = await apiService.createCategory(categoryData);
      console.log('Category created:', response);
    } catch (error) {
      console.error('Error creating category:', error);
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <div>
        <label>Name:</label>
        <div>
          <input
            type="text"
            placeholder="English Name"
            value={categoryData.name.en}
            onChange={(e) => handleFieldChange('name', 'en', e.target.value)}
          />
          <input
            type="text"
            placeholder="Arabic Name"
            value={categoryData.name.ar}
            onChange={(e) => handleFieldChange('name', 'ar', e.target.value)}
          />
        </div>
      </div>

      <div>
        <label>Description:</label>
        <div>
          <textarea
            placeholder="English Description"
            value={categoryData.description.en}
            onChange={(e) => handleFieldChange('description', 'en',
e.target.value)}
          />
          <textarea
            placeholder="Arabic Description"
            value={categoryData.description.ar}
            onChange={(e) => handleFieldChange('description', 'ar',
e.target.value)}
          />
        </div>
      </div>

      <button type="submit">Create Category</button>
    </form>
  );
};

export default CategoryForm;
```

### 4. Review Form

```
// components/ReviewForm.js
import React, { useState } from 'react';
import { useLanguage } from '../contexts/LanguageContext';
import apiService from '../services/api';

const ReviewForm = ({ productId }) => {
  const { currentLanguage } = useLanguage();
  const [reviewData, setReviewData] = useState({
    rating: 5,
    comment: { en: '', ar: '' }
  });

  const handleCommentChange = (language, value) => {
    setReviewData(prev => ({
      ...prev,
      comment: { ...prev.comment, [language]: value }
    }));
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const response = await apiService.createReview(productId, reviewData);
      console.log('Review created:', response);
    } catch (error) {
      console.error('Error creating review:', error);
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <div>
        <label>Rating:</label>
        <select
          value={reviewData.rating}
          onChange={(e) => setReviewData(prev => ({ ...prev, rating:
parseInt(e.target.value) }))}
        >
          <option value={1}>1 Star</option>
          <option value={2}>2 Stars</option>
          <option value={3}>3 Stars</option>
          <option value={4}>4 Stars</option>
          <option value={5}>5 Stars</option>
        </select>
      </div>

      <div>
        <label>Comment:</label>
        <div>
          <textarea
            placeholder="English Comment"
            value={reviewData.comment.en}
            onChange={(e) => handleCommentChange('en', e.target.value)}
          />
          <textarea
            placeholder="Arabic Comment"
            value={reviewData.comment.ar}
            onChange={(e) => handleCommentChange('ar', e.target.value)}
          />
        </div>
      </div>

      <button type="submit">Submit Review</button>
    </form>
  );
};

export default ReviewForm;
```

## Display Components

### 1. Product Display

```
// components/ProductDisplay.js
import React from 'react';
import { useLanguage } from '../contexts/LanguageContext';

const ProductDisplay = ({ product }) => {
  const { currentLanguage } = useLanguage();

  const getLocalizedText = (bilingualField) => {
    if (!bilingualField) return '';
    return bilingualField[currentLanguage] || bilingualField.en || '';
  };

  return (
    <div className="product-card">
      <h3>{getLocalizedText(product.name)}</h3>
      <p><strong>Category:</strong> {getLocalizedText(product.category)}</p>
      <p>{getLocalizedText(product.description)}</p>
      {product.unit && <p><strong>Unit:</strong> {getLocalizedText(product.unit)}
</p>}

      {/* Custom Fields */}
      {product.customFields && product.customFields.map((field, index) => (
        <div key={index}>
          <strong>{getLocalizedText(field.key)}:</strong>
{getLocalizedText(field.value)}
        </div>
      ))}
    </div>
  );
};

export default ProductDisplay;
```

### 2. Address Display

```
// components/AddressDisplay.js
import React from 'react';
import { useLanguage } from '../contexts/LanguageContext';

const AddressDisplay = ({ address }) => {
  const { currentLanguage } = useLanguage();

  const getLocalizedText = (bilingualField) => {
    if (!bilingualField) return '';
    return bilingualField[currentLanguage] || bilingualField.en || '';
  };

  return (
    <div className="address-card">
      <p>{getLocalizedText(address.addressLine1)}</p>
      {address.addressLine2 && <p>{getLocalizedText(address.addressLine2)}</p>}
      <p>{getLocalizedText(address.city)}, {getLocalizedText(address.state)}</p>
      <p>{address.postalCode}</p>
      <p>{getLocalizedText(address.country)}</p>
    </div>
  );
};

export default AddressDisplay;
```

### 3. Category Display

```
// components/CategoryDisplay.js
import React from 'react';
import { useLanguage } from '../contexts/LanguageContext';
```

```
const CategoryDisplay = ({ category }) => {
  const { currentLanguage } = useLanguage();

  const getLocalizedText = (bilingualField) => {
    if (!bilingualField) return '';
    return bilingualField[currentLanguage] || bilingualField.en || '';
  };

  return (
    <div className="category-card">
      <h4>{getLocalizedText(category.name)}</h4>
      {category.description && <p>{getLocalizedText(category.description)}</p>}
    </div>
  );
};

export default CategoryDisplay;
```

### 4. Order Display

```
// components/OrderDisplay.js
import React from 'react';
import { useLanguage } from '../contexts/LanguageContext';

const OrderDisplay = ({ order }) => {
  const { currentLanguage } = useLanguage();

  // Order status is already localized by the API
  // The API returns the status in the user's preferred language
  return (
    <div className="order-card">
      <h4>Order #{order._id}</h4>
      <p><strong>Status:</strong> {order.status}</p>
      <p><strong>Items:</strong> {order.items.length}</p>
      <p><strong>Created:</strong> {new
Date(order.createdAt).toLocaleDateString()}</p>

      {/* Status Log */}
      {order.statusLog && order.statusLog.length > 0 && (
        <div>
          <h5>Status History:</h5>
          {order.statusLog.map((log, index) => (
            <div key={index}>
              <span>{log.status}</span>
              <small> - {new Date(log.timestamp).toLocaleDateString()}</small>
            </div>
          ))}
        </div>
      )}
    </div>
  );
};

export default OrderDisplay;
```

5. Review Display

```
// components/ReviewDisplay.js
import React from 'react';
import { useLanguage } from '../contexts/LanguageContext';

const ReviewDisplay = ({ review }) => {
  const { currentLanguage } = useLanguage();

  const getLocalizedText = (bilingualField) => {
    if (!bilingualField) return '';
    return bilingualField[currentLanguage] || bilingualField.en || '';
  };

  return (
    <div className="review-card">
      <div className="rating">
        {[...Array(5)].map((_, i) => (
          <span key={i} className={i < review.rating ? 'star filled' : 'star'}>
            ★
          </span>
        ))}
      </div>
      {review.comment && <p>{getLocalizedText(review.comment)}</p>}
      <small>By: {review.user?.firstname} {review.user?.lastname}</small>
    </div>
  );
};

export default ReviewDisplay;
```

## Usage Example

```
// App.js
import React from 'react';
import { LanguageProvider } from './contexts/LanguageContext';
import LanguageToggle from './components/LanguageToggle';
import ProductForm from './components/ProductForm';
import ProductDisplay from './components/ProductDisplay';
import AddressForm from './components/AddressForm';
import CategoryForm from './components/CategoryForm';
import ReviewForm from './components/ReviewForm';

function App() {
  return (
    <LanguageProvider>
      <div className="App">
        <header>
          <h1>Bilingual App</h1>
          <LanguageToggle />
        </header>

        <main>
          <section>
            <h2>Create Product</h2>
            <ProductForm />
          </section>

          <section>
            <h2>Create Address</h2>
            <AddressForm />
          </section>

          <section>
            <h2>Create Category</h2>
            <CategoryForm />
          </section>

          <section>
            <h2>Add Review</h2>
            <ReviewForm productId="product-id-here" />
          </section>
        </main>
      </div>
    </LanguageProvider>
  );
}

export default App;
```

## Backend Changes Made

### 1. Updated Models

- **Product Model**: `category` and `unit` fields now support bilingual content
- **Address Model**: `addressLine1`, `addressLine2`, `city`, `state`, `country` fields now support bilingual content
- **Category Model**: `name` and `description` fields now support bilingual content
- **Review Model**: `comment` field now supports bilingual content
- **Order Model**: Status is localized in responses (not stored as bilingual)

### 2. Updated Controllers

- All controllers now support `?lang=en`, `?lang=ar`, and `?lang=both` query parameters
- Added validation for bilingual fields requiring both English and Arabic content
- Added helper functions to format responses with localized content

### 3. Updated Messages

- Added new bilingual validation messages for all new bilingual fields
- All error messages support both English and Arabic

### 4. Updated API Documentation

- Added language support sections for all routes
- Updated object definitions to show bilingual field structure
- Added validation rules for bilingual fields

## Key Features

1. **Language Toggle**: Users can switch between English and Arabic
2. **Bilingual Forms**: All forms support input in both languages
3. **Localized Display**: Content is displayed in the user's selected language
4. **API Integration**: Seamless integration with the bilingual API
5. **Validation**: Frontend validation for bilingual field requirements
6. **Responsive Design**: Forms and displays work well on all devices

This integration guide provides a complete solution for building a bilingual React frontend that works seamlessly with the updated API.